Universität Karlsruhe (TH)

Lehrstuhl für Programmierparadigmen

Sprachtechnologie und Compiler WS 2009/2010 http://pp.info.uni-karlsruhe.de/
Dozent: Prof. Dr.-Ing. G. Snelting snelting@ipd.info.uni-karlsruhe.de
Übungsleiter: Sebastian Buchwald sebastian.buchwald@kit.edu

Übungsblatt 6 | Ausgabe: 26.11.2009 | Besprechung: 02.12.2009

Aufgabe 1: *Earley Parser*

Gegeben folgende Grammatik:

$$
\begin{array}{rcl}
T & \rightarrow & T < - T \\
 & | & T - > T \\
 & | & \& \; T \\
 & | & \textbf{id}
\end{array}
$$

Konstruieren Sie mit Hilfe des Earley Algorithmus alle möglichen Parsebäume für folgende Sätze:

- **id -> id <- id**
- **& id <- id**

Aufgabe 2: *Auswertung Boolescher Ausdrücke*

Boolesche Ausdrücke können oft mit bedingten Sprüngen ausgewertet werden. Dabei muss jeweils die Adresse spezifiziert werden, an der die Berechnung nach dem Sprung fortgesetzt wird. Gegeben sei folgende Syntax Boolescher Ausdrücke:

$$
\begin{array}{rcl}
\textit{conditional\_clause} & \rightarrow & \textbf{if } \textit{boolean\_expr} \textbf{ then } \textit{stmt\_list} \textbf{ else } \textit{stmt\_list} \textbf{ end} \\
\textit{boolean\_expr} & \rightarrow & \textit{boolean\_expr } \textit{boolean\_op } \textit{boolean\_expr} \\
\textit{boolean\_expr} & \rightarrow & \textbf{not } \textit{boolean\_expr} \\
\textit{boolean\_op} & \rightarrow & \textbf{and} \mid \textbf{or}
\end{array}
$$

Die Codeerzeugung soll aus dem AST erzeugen. Dafür müssen die Knoten mit Sprungmarken versehen werden, die mit bedingten Sprüngen angesprungen werden können. Die Funktion new_label erzeugt eine neue Sprungmarke. Das Attribut *location* gibt die Sprungmarke eines Knotens an. Die Attribute *jump_true* und *jump_false* die Sprungziele bei positiver bzw. negativer Auswertung eines Ausdrucks.

Geben Sie eine attributierte Grammatik an, die die Auswertung Boolescher Ausdrücke spezifiziert.

Aufgabe 3: *Attributierte Grammatiken*

Gegeben folgender Auszug aus der Grammatik einer Java-ähnlichen Sprache:

$$
\begin{array}{rcl}
\mathit{CompilationUnit} & \rightarrow & \mathit{Class}\ \mathit{CompilationUnit} \mid \varepsilon \\
\mathit{Class} & \rightarrow & \textbf{class id}\ \{\ \mathit{ClassMembers}\ \} \\
\mathit{ClassMembers} & \rightarrow & \mathit{ClassMember}\ \mathit{ClassMembers} \mid \varepsilon \\
\mathit{ClassMember} & \rightarrow & \mathit{Field} \mid \mathit{Method} \\
\mathit{Field} & \rightarrow & \mathit{Type}\ \textbf{id}\ ; \\
\mathit{Method} & \rightarrow & \mathit{Type}\ \textbf{id}\ \mathit{CompoundStatement} \\
\mathit{CompoundStatement} & \rightarrow & \{\ \mathit{StatementList}\ \} \\
\mathit{StatementList} & \rightarrow & \mathit{Statement}\ \mathit{StatementList} \mid \varepsilon \\
\mathit{Statement} & \rightarrow & \mathit{CompoundStatement} \mid ; \\
\mathit{Type} & \rightarrow & \textbf{void}
\end{array}
$$

Weiter seien Funktionen mit folgenden Signaturen zum Aufbau eines AST gegeben:

$$
\begin{array}{rcl}
\text{newCompilationUnit} & : & \mathcal{P}(\mathit{Class}) \rightarrow \mathit{CompilationUnit} \\
\text{newClass} & : & \mathit{Symbol} \times \mathcal{P}(\mathit{ClassMember}) \rightarrow \mathit{Class} \\
\text{newField} & : & \mathit{Symbol} \times \mathit{Type} \rightarrow \mathit{ClassMember} \\
\text{newMethod} & : & \mathit{Symbol} \times \mathit{Type} \times \mathit{Statement} \rightarrow \mathit{ClassMember} \\
\text{newCompoundStatement} & : & \mathcal{P}(\mathit{Statement}) \rightarrow \mathit{Statement} \\
\text{emptyStatement} & : & \mathit{Statement} \\
\text{voidType} & : & \mathit{Type}
\end{array}
$$

Das Terminal **id** besitzt ein vordefiniertes Attribut $\mathit{symbol} : \mathit{Symbol}$.

Stellen Sie Attributierungsregeln auf, die zu einem Programm einen passendes AST erzeugen.